Procedural Zelda

Heijne, Norbert; Bakkes, Sander

Published in:
International Conference on the Foundations of Digital Games (FDG'17)

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Procedural Zelda: A PCG Environment for Player Experience Research

Norbert Heijne
University of Amsterdam
norbert.heijne@gmail.com

Sander Bakkes
Tilburg University
s.c.j.bakkes@uvt.nl

ABSTRACT

To contribute to the domain of player experience research, this paper presents a new PCG environment with a relatively wide expressive range that builds upon the iconic The Legend of Zelda: A Link to the Past action-RPG game; it contributes by providing the openly-available *Procedural Zelda* environment for gaming research. The paper presents the design goals and design context of the research environment, and provides a detailed overview of the procedural capabilities of Procedural Zelda, together with its capabilities for data logging, to benefit, e.g., player modelling investigations.

CCS CONCEPTS

• **Applied computing** → **Computer games**; • **Software and its engineering** → **Interactive games**; • **Information systems** → *Multimedia content creation*; • **Human-centered computing** → *Human computer interaction (HCI)*;

KEYWORDS

Zelda, Procedural Content Generation, Player experience research, Player modelling

ACM Reference format:
Norbert Heijne and Sander Bakkes. 2017. Procedural Zelda: A PCG Environment for Player Experience Research. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 10 pages.
https://doi.org/10.1145/3102071.3102091

1 INTRODUCTION

An evident benefit of procedural content generation (PCG) – the algorithmic creation of game content with limited or indirect user input [26] – is that it can be employed for the automated generation of game levels. Also, it is interesting to observe that the field has become increasingly more aligned with game research that concerns player (experience) modelling (e.g., [17, 18, 23, 24]), adaptive game experiences (e.g., [2, 3, 15, 27, 28]), game analysis (e.g., [8, 9, 11, 20]), personality analysis (e.g., [4, 5]), mixed-initiative content creation (e.g., [14, 22]), etc. etc.

A central concept regarding PCG-based game environments for research, is *expressive range*. The concept refers to the space of potential levels that a generator is capable of creating, including how biased it is towards creating particular kinds of content in that space [21]. Indeed, the expressive range of a PCG-based game environment is essential for research that is focused on understanding the links between exhibited in-game behaviour, and factors such as player personality, self-reported game experience, player characteristics and preferences, or game literacy. That is, it is desirable that procedural techniques can generate a game environment that allows – and can control for – a wide range of behavioural possibilities for the players, such that distinct players can interact with the game environment in their own, unique play style.

As such, to contribute to the domain of player experience research, this paper presents a new PCG environment with relatively wide expressive range that builds upon the iconic The Legend of Zelda: A Link to the Past action-RPG game; it contributes by providing the openly-available *Procedural Zelda* environment for gaming research

2 DESIGN GOALS

To enable Procedural Zelda to serve as an environment for player experience research, it is designed around the following high-level goals:

(1) Procedural Zelda provides an interactive environment of *relatively wide expressive range* in which distinct players have the opportunity to act in their own, unique play style.[1]
(2) Procedural Zelda provides an investigator with (i) *offline control* over important facets of the procedural game design (i.e., before a game is started), such that a game may be generated that is tailored to an individual player, and (ii) *online control* (i.e., during gameplay), such that important aspects of the running game can be adapted / can be (re)generated on the fly.
(3) Procedural Zelda provides *data logging capabilities* that enable (i) monitoring (and modelling) of numerous facets of observed player behaviour, (ii) querying the player for self-reported game experiences during and after gameplay.

To facilitate these high-level design goals, Procedural Zelda generates a complete quest that lasts 20 to 60 minutes (depending on play style) in which the game takes place. It includes a handcrafted Village with non-player characters (NPCs) who introduce the storyline of the quest to the player, a Tutorial level, a Shop, numerous procedurally generated Levels, an adaptive Boss Fight, and a conclusion to the quest (this is further detailed in Section 4.1). A wide

[1] We consider the relatively wide expressive range as resulting from (i) the relative complexity of Zelda's gameplay itself, (ii) the generative control over the world generation (discussed next), and (iii) how versatile the environment is for offering player behaviour choice. Aside these general observations, a more formal analysis of the generator's expressive range is relevant and interesting too, but falls outside of the scope of the present paper.

FDG'17, August 14-17, 2017, Hyannis, MA, USA

ACM ISBN 978-1-4503-5319-9/17/08…$15.00
https://doi.org/10.1145/3102071.3102091

Table 1: Overview of game content in Procedural Zelda. A list of adjustable parameters is available at https://tiu.nu/pz-appendix

Element	Description	Handcr.	Offline gen.	Online gen.
Village, Tutorial, Shop	Game assets, general design, NPCs, storyline	✓		
Levels	Mission & space generation, branching & path lengths, asset placem., combat & puzzle placem.		✓	
Combat encounters	Encounter staging, difficulty adjustments, resource drops, boss fights			✓
Puzzles	Puzzle complexity of Maze, Moving floor, and Sokoban puzzles			✓

variety of procedural elements allows the investigator to make numerous parts of the game responsive to game metrics (a) at design time, before the game is being started, and (b) at run time, to adapt the elements in response to player actions (Table 1, this is further detailed in Section 4.2 – 4.6). The provided data logging routines are further detailed in Section 5.

3 DESIGN CONTEXT

Procedural Zelda purposely builds upon design conventions and assets from the game *The Legend of Zelda: A link to the past*; an iconic action-RPG game that provides rich opportunities for players to express themselves in distinct manners. The player is confronted with ever-increasing challenges throughout the game that require either reflexes and hand-eye coordination or solving a puzzle with logic and wit.

The game carries conventions that have been very successful within the action-RPG genre; the most prevalent being:

- Combat is done with some kind of weapon (often close range), with the player having only a rudimentary control over the weapon (a button causes a specific attack animation). The player then complements the rudimentary combat with tools and items found in the world.
- The player can track statistics of, e.g., the current strength of the character (life points displayed as hearts and magic power displayed as a green bar, see Figure 1).
- When the character has zero life points left it is game over, gets stronger as the game progresses, either through an increase in statistics or available tools.
- The world contains secrets to find which are often unlocked through the use of an item, tool, solving of a puzzle or defeat of a stronger than usual enemy.
- The story often sends the hero to a dungeon (an enclosed area) which is filled with enemies and traps and ends in a climactic boss fight (a much stronger enemy which often requires the use of the item found in the dungeon to defeat it).
- Characters within the story often give hints on how to play the game or how to proceed with the story.

Figure 1: Procedural Zelda builds upon the conventions and assets of the game The Legend of Zelda: A Link to the Past (illustrated). From top-left to bottom-right: fighting on a bridge, walking through the village, receiving an important item, fighting a boss.

Procedural Zelda builds upon these same game mechanics and conventions, albeit that we will generate our levels algorithmically, and the scope of the game is naturally less extensive (it provides 20 to 60 minutes of gameplay, depending on the play-style that is adopted by the user). Indeed, we expect researchers to be readily able to investigate facets of player experience, when the research environment builds upon proven game conventions and upon assets (e.g. characters, tile sets, enemies) that are already deemed effective. *Zelda* games often contains combat, puzzles, story elements, exploration and is an iconic title that most gamers would recognize. As such, it enables many distinct opportunities for interaction which, particularly when compared to 2D platform games, provides a wide scope for possible gaming investigations and provides rich potential for data gathering and player modelling.

4 PROCEDURAL ZELDA

We provide the Procedural Zelda game environment via GitHub.[2] The game environment builds upon Solarus [25], an open-source game engine that has implemented its own version of *The Legend of Zelda: A Link to the Past*. The engine has a relatively low footprint in terms of processing power, and already implements many gaming assets and conventions that players will be familiar with. Procedural Zelda has already been employed for investigating the correlation of five-factor model personality traits, in-game observations, and self-reported game experiences, of which the first experimental finding will be reported in a separate article.

[2]Procedural Zelda is available online at https://github.com/zeldaresearch/Zelda-ALTTP-Platform. A demo video of research using Procedural Zelda is available online at https://tiu.nu/procedural-zelda.

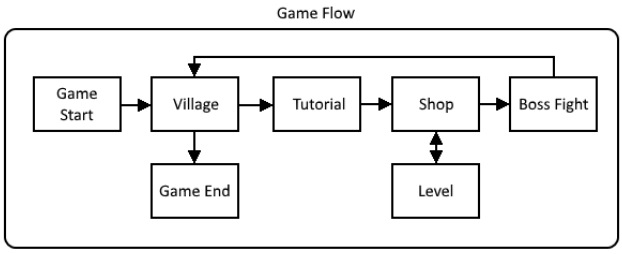

Figure 2: Flow of the game.

We will subsequently discuss in detail the general structure of the game (4.1), and the numerous procedural elements that it contains, being a village (4.2), and components for level generation (4.3), combat generation (4.4), puzzle generation (4.5), and exploration rewards and an in-game shop (4.6). Note that numerous descriptions are purposely detailed, and – as the described elements generally are procedural in nature – they are intended to convey an intuition of Procedural Zelda's expressive capabilities.

4.1 General structure of the game

The game's structure is set up as followed:

- The Village: A place where the player is introduced to the quest, has the opportunity to learn more about the background of the characters and prepare for the journey ahead.
- Tutorial Level: A level with no exploration options or extra resources, it contains four combat rooms and one of each puzzle at the lowest difficulty.
- Shop: The shop houses a choice of equipment pieces that could simplify certain tasks, primarily tasks that involve getting hurt such as the mazes, the moving floor rooms or the fights. These pieces can be bought with money found in the levels. Exiting the shop transports the player to the next level.
- Levels: A themed segment is made up out two levels with a generated layout in a particular theme that contains rooms that either generate combat, puzzles or a treasure chest. This is the main part of the game. After completing a themed segment, the player is transported to the shop. There are three segments in total.
- Boss fight: A typical boss fight for a *Zelda* game, this is handled by our combat generator.
- The Village again: The player finishes up the quest with a choice on how to fulfil the quest requirements and completes the game.

4.2 Village

The handcrafted Village (Figure 3) is where the player starts his adventure, the player is spawned near A2 with sinister music playing in the background. The player is now motivated by the player character's father (A2) to go and seek help from the witch (C1) to cure the player character's brother (A1) from his illness and is given the instruction to head east to ask her for help after picking up the Sword and Shield from the shed.

Some of the NPCs are placed specifically to allow players to prepare for the oncoming journey. These include the shopkeeper

Figure 3: The Village.

(B1) which sells apples to the player, which restores 1 heart of the players health pool. The boy who lost his milk bottle (B8) reveals that he lost his bottle and asks the player to find it among the bushes. The player gets to keep the bottle when he finds it, which can later be used to fill it with potion from the witch to attain background information on the witch and to contain fairies which revive the player on death.

To complete the game after acquiring the "Cure Flower" quest item, the player has to talk to the brewer (B6) or the witch (C1) to make the cure. The brewer will do this for free, and create a strong cure. The witch will ask the player for 50 rupees (i.e. the coins used in Zelda) and creates a diluted cure. Both will cure the illness of the brother, thus the choice is there for flavor. When the player presents the cure to A1 the game's credits roll and the game is complete. All other NPCs are non-beneficial and only add flavor and background information to the game.

The Village is built in such a way that little interaction is required and that conversational content can be ignored if the player wishes to follow the instructions as fast as possible.

4.3 Level generation

In this section we will present all the parameters used when generating a level and how a level is generated. The levels are built in four steps:

(1) A level's content is first generated as a graph representation which is called a mission. The mission only has information on the connectivity of rooms, their type and the content of the chests. The generation process is based on the *mission generation* process described in [1, 7].
(2) Spatial planning is laid out using the *maze generator* that prioritizes placing rooms such that the level always stretches outward. Pathways inside the rooms are also laid out using the maze generator.
(3) Assets are placed; for the forest and swamp levels the tree background is generated, for the cave the walls, floors and doorways are generated. Afterwards any space left that was designated as non-walkable is filled with static props, throwable props (bushes or rocks) or hazardous tiles (such as water or holes).

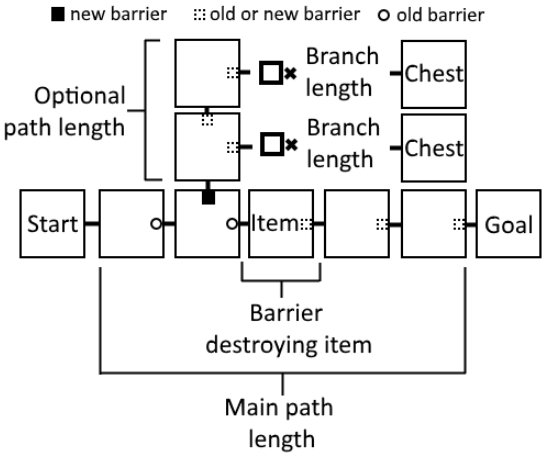

Figure 4: Level layout and use of the level length parameters for generating a level

Table 2: The length parameters (amount of rooms) for generating a level.

Level name	Branch length	Optional path length	Main path length	Barrier type
Tutorial	N/A	0	7	bush
Forest 1st lvl	1	2	4	rock
Forest 2nd lvl	2	2	4	rock
Swamp 1st lvl	3	2	4	large skulls
Swamp 2nd lvl	4	2	4	large skulls
Caves 1st lvl	5	2	4	bomb-able blcks
Caves 2nd lvl	6	2	4	bomb-able blcks

(4) Barriers are placed, such as bushes and rocks that block specific entrances. Treasure chests are placed.

4.3.1 Level length settings. The procedural process for determining the level layout is illustrated in Figure 4; the parameters used in generating a level can be found in Table 2. A themed segment (Forest / Swamp / Caves) consists of two levels. A level has a main length of rooms which are either a puzzle or fight, added to that is a start and an end room, and the first level of a segment has a room with a treasure chest halfway through the main path that holds an item with which the player can destroy the new type of barriers in that segment. Each themed segment has a unique type of barrier that is being placed.

An optional path entrance is placed just before the treasure room or the halfway point that is blocked by a new type barrier, requiring the player to go back in the first level of a segment if they want to explore further. Each room in the optional path opens up a path to a branch that has a specific number of rooms before entering a room with a treasure chest that holds 100 rupees. In the second level one of the optional treasure chests contains a heart container.

The purpose of the linearly increasing branch lengths is to increase the time between the payout of the extrinsic rewards and to avoid a ceiling effect when it comes to the number of explored optional rooms.

4.3.2 Mission generator. The underlying mission generator is used primarily to generate a graph that can be used within our generation process. It uses the parameters and generation process described in the previous section. During development the generic implementation of the mission generation found in the paper [7] was adapted such that one can specify a branching-factor to determine how unique / static the maps should be; to determine if the generation of maps outside of specific measurement bounds is allowed.[3]

The mission generator still uses the same principle as described in the paper; a graph representation is used with directional and bidirectional relations, and the nodes in combination with the relation determine what the node represents in our generation process. This allows us – if desired – to avoid stringing together multiple of the same tasks after one another (e.g., alternate between fight and puzzle) such that the player is predictably not bored with too much repetitiveness. And, in addition, it allows that the main path can always contain the same amount of fights and puzzles, such that one will get about as many encounters for fights and puzzles; during a level with a specific difficulty in the case of static difficulty this can be particularly important because it allows resulting encounter data to be comparable between levels (if desired).

Once the mission generator parses the graph representation one obtains a mission representation, which contains the data for rooms and their connection to other rooms, barrier placement, for chest rooms their chest contents, start and exit rooms as well as the overall level information about difficulty, theme and type of map (tutorial, normal or boss). This data is sent to the space generator to generate the level layout.

4.3.3 Space generator. Our space generator can be considered a maze generator to produce the layout of our level; it generates the actual map. The maze generator takes in three parameters: wall width (x pixels by y pixels), corridor width (x pixels by y pixels) and total maze measurements (x corridors by y corridors) (illustrated in Figure 5). The maze generator provides us with a grid where each node contains information on which wall is open, closed or contains a small pathway to the other node.

Starting with the entrance area at the most western room in the middle of the "maze", it assigns the room its number that is associated with a room in the generated mission representation, and it assigns the connected rooms to the adjacent rooms that have the most space left (a random choice of a room with the most adjacent unassigned rooms in two degrees, see Figure 6). Connections to other rooms are designated as a small pathway, if a room is larger than one node the walls between the nodes are designated as open.

So-called fight rooms are square and take up the size of one room, rooms for puzzles need the extra space to be viable as puzzles and take up the size of two rooms. Boss rooms are also square but take up four rooms.

[3]Cf. [7], a replacement-grammar is employed for level generation, which can be expanded to adapt the level's structure during actual gameplay.

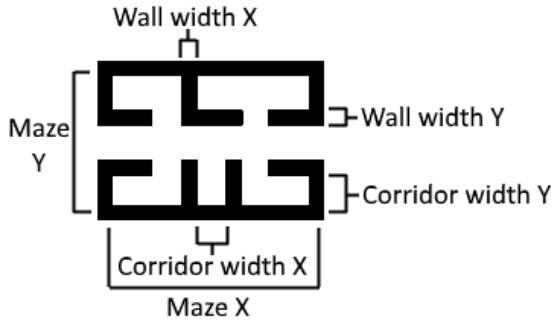

Figure 5: Visual representation of the maze generation parameters

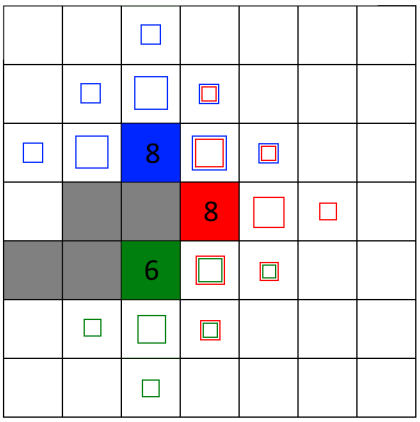

Figure 6: Placing a room within a grid. Grey is an already present room. Green, Red and blue indicate possible locations, the colored rectangles signify the unassigned rooms in two degrees.

We now have a grid as our spatial representation with which to further improve our layout. The next step involves carving paths inside the non-puzzle rooms (puzzle rooms need all the space available). For this we use the maze generator once more. We first create a grid of the room with the corridor size of 16 pixels by 16 pixels (the size required for our character to be able to walk through), we then subdivide our room over the length and width into sections, and use the center cells (with a random offset) of each section to use our maze generation algorithm on. Using the smaller section representation we use path-finding to open up cells from each center cell to another cell where the section representation indicated that there is an opening. The opened cells are widened resulting in a pathway that connects the entrance with all exits in the room, the still unopened cells in the grid are used for prop placement.

4.3.4 Placement of assets. The assets that we use are made up of sprites and tiles. Sprites are objects in the game that have an animation based on their internal state that can be interacted with by input of the player. Tiles are static and are placed on a layer. Tiles cannot be interacted with but can affect the player (water, holes, spiked floor all cause damage); they can be arranged in a pattern to form a prop (for example a tree is made of tiles that are placed on multiple layers in a pattern). We handcrafted many of these props to be used together with each theme. The produced spatial representation of the previous section is used to fill the level with sprites, tiles and props.

Figure 7: Themes: Top-left, Caves. Top-right, Swamp. Bottom, Forest.

Procedural Zelda comes with assets for three distinct themes: caves, swamp, and forest (illustrated in Figure 7). In case of the Forest and Swamp themes, the walls and non-walkable sections are filled with trees that are placed in a repeating pattern that is common in the *The Legend of Zelda: A Link to the Past* game. These trees are only placed if there is space to place an entire tree. The remaining non-walkable space in the rooms is filled randomly with either water (the player jumps a certain distance when touching the edge of the water), throwable sprites (bushes or rocks), smaller props or designated as empty space.

In case of the Caves theme, the wall props are placed and remaining space in the walls section is filled with a specific tile, which is also common for *The Legend of Zelda: A Link to the Past*. The non-walkable space in rooms is filled randomly with either spiked flooring (which damages and stops player movement), pits (where the player directly falls into, receives damage and is placed back on the last spot where the player was safely standing), throwable sprites (mostly rocks), or smaller props.

These changes in asset placement can be used to linearly increase difficulty besides the already present adaptable difficulty, as well as to provide a noticeable change in the way one can move around in the environment.

4.4 Combat generation

Combat generation is composed of staging the encounter (4.4.1), determining resource drops (4.4.2), and optionally setting up a boss fight (4.4.3).

4.4.1 Encounter staging. Encounter staging is provided via two implementations that generate enemy encounters; one with static

Figure 8: The enemies: From left to right, Hardhat, Snapdragon, Minillosaur, Green knight.

Table 3: The settings for static combat, H: Hardhat, S: Snapdragon, M: Minillosaur, G: Green knight.

Level number	Enemy spawn combinations
Tutorial	[M×4] OR [G×2] OR [S×3]
Forest	[M×5] OR [G×3] OR [S×5]
Swamp	[M×7] OR [G×4] OR [S×6]
Caves	[M×8 + H×2] OR [G×4 + H×2] OR [S×6 + H×2] OR [H×5]

linearly scaling difficulty, and one with adaptable difficulty. Both implementations use a specific set of enemies, illustrated in Figure 8.

The enemies each have their own behavior pattern as well as some shared behavior:

- The enemy types Hardhat, Minillosaur and Green knight moves towards the player once it is within 100 pixels of the player.
- The player loses half a heart when the player's character touches an enemy.
- The enemy types Hardhat, Snapdragon and Green knight move about randomly before spotting the player.
- The Hardhat repels the player backward when it is hit with a sword strike.
- The Minillosaur starts out as an immobile egg, and cracks open once the player is within range.
- The Snapdragon always moves about randomly in a diagonal pattern and ignores the player.
- The Green Knight's sword repels the player's sword attacks, preventing damage but repels the Green Knight backwards.

The provided static difficulty is handcrafted and linearly increases between levels. Hardhat enemies only spawn in the caves on this difficulty setting, and the amount of enemies spawned are based on observed play-tests of "average" players.

The adaptable difficulty is made with flow channels loosely in mind [6, 13]; it balances the enemy composition in a room in such a way that it attempts to find the optimal weight per enemy type based on the player's performance and actions using *linear regression*. The combat's difficulty is subsequently adjusted upward and downward in a sinus wave pattern such that the highest difficulty is the optimal difficulty given the player (i.e., a difficulty that the player would just about handle).

One may expect that (hardcore) players that enjoy challenges would prefer the adaptable difficulty over the static difficulty because it presents a more suited challenge a lot earlier than the static difficulty, and may expect that casual players will enjoy relatively short combat with low difficulty and therefore would prefer the static difficulty in the earlier levels.

Table 4: The loot table used for enemies and picking up throwable props.

Type of drop	Hearts	Arrows	Bombs	Magic	Rupee
Amount	1	2	1	1	4

4.4.2 Resource drops. To decrease the amount of noise in the difficulty of the combat we implemented so-called resource drops in a deterministic way. That is, the amount of hearts available to the player will determine the amount of mistakes the player can make and consequently the difficulty of the fight. If some fights would grant a lot of hearts and others none then it would be detrimental to our data. As such the loot table (see Table 4 for details) is determined beforehand, an item is dropped from the loot table randomly with no returns whenever an enemy is killed or a throwable prop is picked up. When the loot table is empty it is refilled.

4.4.3 Boss fights. The 'boss' enemy always has a static difficulty (in the provided implementation, it requires 6 hits with bomb explosions) and is relatively easy as long as the player has the convention of looking for a monster's weakness, pays attention to auditory (any hit with a sword bounces off) and visual cues (small minions drop bombs) and reads the bomb bag acquisition description that the bombs are throwable or knows the *Zelda* convention that the boss is always beaten with the use of the last equipment piece found in the dungeon.

Generally speaking, the qualities that would help improve the player's performance with combat, are reaction time and planning. Timing sword swings decreases the difficulty of the Green knight enemies the most and is useful against all enemy types. Hardhats are more easily dealt with using items or throwable props. Using any type of item from the shop will decrease the difficulty of the fights. Timing the sword swings would indicate a larger precision and less sword swings overall.

4.5 Puzzle generation

A puzzle is generated when the player enters a room that has been designated as a puzzle room by the mission generator. Procedural Zelda provides three distinct procedurally generated puzzle types: a maze puzzle (4.5.1), a moving floor puzzle (4.5.2), and a Sokoban puzzle (4.5.3).

In the provided implementation, which puzzle type is generated is based on the amount of puzzles of each type that has been generated, the type is then chosen randomly between the types that have been generated the least amount of times such that the generated amount remains the same but with random ordering. As with combat generation, we implemented two variants of the difficulty, a static linearly scaling difficulty and an adaptable version. The adaptable version of the puzzle generation carries a simpler version than the combat generation. That is, the difficulty increases or decreases based on the time it took to complete the puzzle, the life lost during the puzzle, or whether the player has died during the puzzle or in the case of the *Sokoban* puzzle type the player pressed

Figure 9: The hazards present in the maze: left to right, fireball statue, bouncing hazard, reactive spike block, pit.

the optional "Quit puzzle" button. Implemented cut-off points have been determined with play-tests of persons with average to high level of skill.

4.5.1 Puzzle type: Maze. Procedural Zelda is able to procedurally generate a maze in the designated room using Prim's algorithm [19]. This process results in a maze without loops with a fixed start- and endpoint and random paths in between with a good amount of branching.

The difficulty of a maze of the above type (no loops, completely random and with fixed start- and endpoints) is usually defined by the size of the maze which comes down to the combined length of paths that are most likely to go towards the endpoint in the maze. A maze is unlikely not to be solved when the player can see the entire maze and without loops in the maze a player can reach the end in a reasonable amount of time with just trial and error. In our game world it is generally not desirable (though not impossible) to make very large mazes as this typically impacts the flow of the game and is dependent on some limitations of the Solarus game engine. As such the following features were added to increase the difficulty of the maze (some are illustrated in Figure 9):

- *Darkness*: the entire maze is covered in darkness and the player can only see a small area ahead of him. The player can use the Magic mirror item at the cost of magic points to illuminate the maze for a short time.
- *Pits*: Some of the dead ends have black pits which cause the player to be transported back to beginning of the maze as well as lose life points.
- *Bouncing hazards* (official enemy name is Bubble): These invulnerable enemies bounce their way through the maze and touching them causes the player to lose life and magic points.
- *Fireball spitting statues*: Another invulnerable enemy which shoots a projectile at the player roughly every four seconds, the player can use his shield to block the projectile or evade it through lateral movement. This projectile moves through walls in the maze.
- *Reactive spike blocks*: Yet another invulnerable enemy which automatically moves at a fast speed towards the player when the player is directly horizontally or vertically aligned with the enemy and within a certain range, the enemy cannot move past the walls of the mazes.

Table 5: Maze puzzle – Generation parameter settings per difficulty type.

Difficulty level	1	2	3	4	5
Fireball statues	0	1	2	3	4
Bouncing hazards	0	1	2	3	4
Pitfalls	No	Yes	Yes	Yes	Yes
Reactive spike blocks	No	No	No	Yes	Yes
Darkness	Yes	Yes	Yes	Yes	Yes

Table 6: Maze puzzle + Moving floor puzzle – Settings of the static difficulty type.

Static	
Level name	Difficulty
Tutorial	2
Forest	3
Swamp	4
Caves	5

Table 7: Maze puzzle + Moving floor puzzle – Settings of the adaptable difficulty type. The amount of hearts lost (HL) or the time spent (TS) versus the cut-off point (COP) is used to determine the change in difficulty.

Adaptable	
Hearts lost AND/OR Time spent (seconds) COP = 20 × ((difficulty +1)/2)	Difficulty change
≤ 1 HL AND TS ≤ COP	+1
≤ 1 HL AND COP < TS ≤ COP ×1.5	+0.5
≤ 2 HL AND COP < TS ≤ COP	+0.5
> 3 HL OR TS > COP ×1.5	-0.5
> 4 HL OR death	-1

Details on provided settings for static and adaptable difficulty can be found in Table 6 and Table 7. The settings for generating a maze puzzle given a certain difficulty level can be found in Table 5.

Generally speaking, the likely qualities that would help in solving this puzzle type is reaction time to avoid spike blocks, pits, projectiles and bouncing hazards and planning to block the projectiles, which would indicate a relatively small amount of life lost and a larger amount of time taken. Reckless behaviour would most likely result in falling in pits, relatively large losses of life, but small amount of time spent on completing the maze.

4.5.2 Puzzle type: Moving floor. The procedurally generated moving floor puzzle is a maze that has large corridors in which the floor moves in a particular pattern and speed, and where touching the wall causes damage to the player. The player can see the movement of the floor and has full visibility of the environment.

Figure 10: Examples of the moving floor puzzle generation within a level. Left has difficulty 3, right has difficulty 5.

Table 8: Moving floor puzzle – Generation parameter settings per difficulty.

Difficulty level	1	2	3	4	5
Corridor width (× character width)	4	3.5	3	2.5	2
Speed	48 pixels per second				
Pattern	Random between up-down / left-right / up-right-down-left				

Also this puzzle can be implemented with a static or an adaptive difficulty level; implementation details of which are provided in Table 6 and Table 7. The settings for generating a moving floor puzzle given a difficulty level can be found in Table 8.

Generally speaking, the likely qualities that would help in solving this type of puzzle is planning and timing: The floor moves in a repeating pattern for a set amount of time at a constant speed with visual acceleration and deceleration, if the player positions the hero properly the player can manoeuvre through the puzzle without harm. However, reckless action by the player will result in constant harm from correcting mistakes while the floor continues to execute its pattern.

4.5.3 Puzzle type: Sokoban. Finally, Procedural Zelda implements *Sokoban* puzzle type. Sokoban is an already existing puzzle type which entails the following: The object of the game is to move blocks on top of switches by pushing them with a character; there exist at least as many movable blocks as there are switches; the player wins when every switch has a block on top of it. The game is heavily affected by the layout of the walkable space and the placement of the movable blocks and switches.

Generating a Sokoban puzzle can be performed via random generated layouts which are then tested for difficulty and feasibility by a bot [16]. Our game however relies on on-the-fly generation of puzzles and as such a generation pipeline would no longer (directly) operate in real-time, though such implementations are possible though we consider them beyond the scope of our research environment. Indeed, there exist many fun handcrafted puzzles which already have a generalized text format and therefore we opted to use an existing, downloaded set of handcrafted layouts [10] that are small enough to fit in our rooms, and for which the content is generated in-game by parsing the adjusted set and placing them in the room at which point our algorithm makes the layout playable.

Figure 11: Example of the Sokoban generation: Layout group 11, 1st in the group, difficulty 3. (#): Wall, (.): Switch, (@): Entrance, (E): Exit, ($): Block, (_): Empty, (R): Teleport.

An example of the layout and details on the handcrafted puzzles can be found in Figure 11. These adjustments have been made to the downloaded set:

- An entrance and exit have been added, the exit is blocked by a wall, which opens up when all switches have blocks on them. All movable blocks and switches also disappear when the player completes the puzzle.
- A switch was added at the entrance to reset the puzzle and teleportation tiles (which teleports the player on top of the reset switch) have been added at multiple spots instead of a wall, such that the player can always reset the puzzle even if the player blocks off the path to the reset switch with a movable block.
- Difficulty annotation has been added to each layout based on play-tests.
- The layouts that are similar and use the same trick to solve the puzzle have been grouped together and ordered in difficulty.

The player must always complete each group of Sokoban puzzles in order of difficulty, and is not suddenly presented with a higher difficulty version while the player hasn't completed the lower difficulty version. The layout used is randomly chosen from the list of layouts of a certain difficulty where the layout is the first in the group, or the player has completed the layouts that came before in the group's ordering. When no layouts fit the criteria the procedural algorithm searches for layouts one level of difficulty lower and fitting the same criteria. If no fitting layouts are found, it selects a difficulty level higher than the current. The adaptable difficulty takes the difference between current difficulty and selected difficulty into account when deciding whether to change the current difficulty setting when the puzzle is completed.

Generally speaking, Sokoban is a type of puzzle that requires the player to spot the trick used to complete the puzzle, this can take a long while depending on the player. Being able to plan ahead is vital in completing these puzzles. We have therefore added a "Quit" option for this puzzle where after a certain time a message is displayed on screen saying that the player can press a button to instantly complete the puzzle. This is an option to ensure that possible research is not hindered by the player's inability to solve a puzzle. The adaptable difficulty and logging both take the quit option into account. The details on the settings for static and adaptable difficulty can be found in Table 9.

Table 9: The Sokoban settings for the adaptable and static difficulty. In case of the adaptable difficulty level is rounded down when used.

Static		Adaptable	
Level name	Difficulty	Time spent (seconds)	Diff. adapt.
Tutorial	2	≤ 90	+1
Forest	3	> 90 and ≤ 135	+0.5
Swamp	4	> 135	-0.5
Caves	5	> 135 & pressed quit	-1

4.6 Exploration rewards and the shop

Most games apply some form of extrinsic rewards to complement the intrinsic rewards of playing, completing a challenge or completing a non-mandatory task. In the Zelda series, a shop was added to give players more choice in their combat play style and to present extrinsic rewards for exploration. In Procedural Zelda, the player has a possible income of at least 300 rupees in between shops, with a bonus on top for killing enemies. All items in the shop cost 300 rupees to buy with the exception of *Apples* (15 for 3) and *Fairies* (50 for one). Furthermore, one of the chests in the second segment in between shops contains a full heart container which restores the player's life points and increases the maximum life points a player can have.

The following purchasable items were added to the shop:

- *Bottle*: An extra bottle to carry an extra fairy, the amount of life the player has available is an indicator for how reckless the player can be before having to restock on life points. The player can only buy one extra bottle.
- *Fairy*: Buying a fairy while the player has an empty bottle results in a bottled fairy which is released upon the players death, restoring the player back to life with full life. If the player has no empty bottles the fairy is instead consumed immediately, restoring the player to full life.
- *Three apples*: The player can own a maximum of 10 apples, and the player buys 3 at a time. An apple can be consumed by the player to restore one heart.
- *Magic mirror*: The player can use this item at the cost of magic points to blind all enemies on the screen, effectively immobilizing them and during this time, contact with the enemies no longer causes damage. The player can also utilize this item to illuminate a dark maze for a short time.
- *Bow*: The player can shoot arrows in a straight line either horizontally or vertically which kills normal enemies if it hits. The player can find arrows by lifting bushes, lifting rocks or killing enemies. The player can hold up to 10 arrows.

The shop is inhabited by an NPC that (when desired by the experimenter) asks the player to fill in an in-game questionnaire about the last themed segment they completed. The NPC, the shop and the in-game questionnaire also purposely act as an interruption of the flow of the game such that the data gathered in the following segment is much less based on the feeling of the segments that came before.

Figure 12: Items in action: Top, bow and arrow. Right, fairy release after death. Left, magic mirror.

Figure 13: The shop.

5 DATA LOGGING CAPABILITIES

Procedural Zelda currently provides data logging functions for 139 observational features. The features pertain specific behavioural traits regarding playing style and player performance. As desired from our own research into correlates of in-game behaviour with player personality or self-reported experiences, numerous features could be considered as being (partially) linked to five-factor model traits regarding openness to experience, conscientiousness in performing gaming tasks, extraversion, and behaviours that may be linked to agreeableness and neuroticism. By providing these logging capabilities, we foremost hope to provide some foundation to diverse gaming investigations.

An overview of features for which data can be stored is given in Table 10.[4] One can observe that in the Village area 19 features are gathered pertaining, among others, extraversion in approaching NPCs and exploring dialogue option, extraversion in discovering the environment, and conscientiousness in performing an assignment. For each Combat encounter detailed data is gathered (54

[4]For an extensive list of the data logging capabilities of Procedural Zelda we refer the reader to [12], pp. 95–102.

Table 10: Overview of data logging capabilities of Procedural Zelda.

Game element	#Fts.	Focus of features
Village area	19	Extraversion in approaching NPCs and exploring dialogue options / Extraversion in discovering the environment / Conscientiousness in collecting distinct elements present in the environment / Conscientiousness in performing an assignment / Time spent in the village
Combat enc.	54	Number and type of enemies present in an encounter / The health and items carried by the player / Specific layout elements present in the scene / Conscientiousness in approaching the encounter / Performance and specific playing style when interacting with the enemies
Puzzle areas	29	Puzzle type and its relative difficulty / Time spend on the puzzle and puzzle elements / Number of retries (where applicable) / Times the player got hurt or made mistakes / Conscientiousness in solving the puzzle (e.g., quitting) / Efficiency in solving the puzzle
General int.	32	Extraversion in exploring the environment / Openness to paths optional to the main path / Number and type of rooms that the player has encountered and completed / Rewards retrieved / Time spent in specific (possibly optional) areas or encounters
Questionnaire	5	Optional likert-scale input on features pertaining exploration / puzzle-solving and preference / combat encounters / overall experience

features for each encounter scene), that do not only focus on combat performance, but also focused on features related to playing style (e.g., avoiding certain enemies). For Puzzle areas diverse data (29 features) is gathered concerning the puzzle solving performance and distinct styles therein (e.g., tendency to quit, mistakes being made). Also for general environment interactions (i.e., not concerning combat encounters and puzzles) data is being regarding on e.g., extraversion in exploring the environment, openness to optional paths, and retrieved rewards (32 features). Finally, for research purposes an optional in-game questionnaire can be queried and store input on the user's game experience (in our implementation, 5 features on player preferences and overall experience).

6 DISCUSSION

This paper presented a new PCG environment with a relatively wide expressive range that builds upon the iconic The Legend of Zelda: A Link to the Past action-RPG game; it contributed the openly-available *Procedural Zelda* environment for gaming research. Procedural Zelda is designed with player experience research in mind, and as such allows the generative game-design to be tailored to external metrics at design time, allows gameplay to be responsive to in-game observations at run-time, and provides data logging capabilities that enable the monitoring (and modelling) of numerous facets of observed player behaviour.

An example of an investigation for which Procedural Zelda has been employed, is a study into the correlation of five-factor model personality traits, in-game observations, and self-reported game experiences. The first experimental findings of this study will be reported in a separate article.

Acknowledgement. Part of the code present in the Procedural Zelda environment is developed by Arjen Swellengebrel, whose efforts are gratefully acknowledged. He is sorely missed.

REFERENCES

[1] Sander Bakkes and Joris Dormans. 2010. Involving Player Experience in Dynamically Generated Missions and Game Spaces. In *Game-On'2010*. 72–79.
[2] Sander Bakkes, Shimon Whiteson, Guangliang Li, George Viorel Vişniuc, Efstathios Charitos, Norbert Heijne, and Arjen Swellengrebel. 2014. Challenge balancing for personalised game spaces. In *IEEE GEM 2014*. 1–8.
[3] Paris Mavromoustakos Blom, Sander Bakkes, Chek Tien Tan, Shimon Whiteson, Diederik M Roijers, Roberto Valenti, and Theo Gevers. 2014. Towards Personalised Gaming via Facial Expression Recognition.. In *AIIDE 2014*.
[4] Alessandro Canossa, Jeremy Badler, MAgy Seif El-NAsr, Stefanie Tignor, and Randy Colvin. 2015. In Your Face(t): Impact of Personality and Context on Gameplay Behavior. In *FDG 2015*.
[5] Alessandro Canossa, Josep B Martinez, and Julian Togelius. 2013. Give me a reason to dig Minecraft and psychology of motivation. In *IEEE CIG 2013*. 1–8.
[6] Ben Cowley, Darryl Charles, Michaela Black, and Ray Hickey. 2008. Toward an understanding of flow in video games. *CIE* 6, 2 (2008), 20.
[7] Joris Dormans and Sander Bakkes. 2011. Generating missions and spaces for adaptable play experiences. *IEEE TCIAIG* 3, 3 (2011), 216–228.
[8] Anders Drachen, Christian Bauckhage, and Christian Thurau. 2015. The Age of Analytics. *IEEE TCIAIG* 7, 3 (2015).
[9] Anders Drachen, Alessandro Canossa, and Georgios N Yannakakis. 2009. Player modeling using self-organization in Tomb Raider: Underworld. In *CIG 2009*. 1–8.
[10] Aymeric du Peloux. 2000. Sokoban handcrafted puzzles: Mini Cosmos. (2000). http://sneezingtiger.com/sokoban/levels/microcosmosText.html.
[11] M Seif El-Nasr, Anders Drachen, and Alessandro Canossa. 2013. Game analytics. *New York, Sprint* (2013).
[12] Norbert Heijne. 2016. Investigating the Relationship between FFM, Game Literacy, Content Generation and Game-play Preference. (2016). UvA, MSc thesis.
[13] Robin Hunicke. 2005. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI ACE 2005*. 429–433.
[14] Antonios Liapis, G Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *FDG 2013*.
[15] Ricardo Lopes, Ken Hilf, Luke Jayapalan, and Rafael Bidarra. 2013. Mobile adaptive procedural content generation. In *Proceedings of the PCG 2013*.
[16] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. 1996. Automatic making of Sokoban problems. In *Pacific Rim Int. Conf. on Artificial Intelligence*. 592–600.
[17] Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N Yannakakis. 2013. Imitating Human Playing Styles in Super Mario Bros. *Ent. Comp.* 4, 2 (2013), 93–104.
[18] Chris Pedersen, Julian Togelius, and Georgios N Yannakakis. 2009. Modeling player experience in super mario bros. In *IEEE CIG 2009*. 132–139.
[19] Robert Clay Prim. 1957. Shortest connection networks and some generalizations. *Bell Labs Technical Journal* 36, 6 (1957), 1389–1401.
[20] Rafet Sifaa, Anders Drachen, and Christian Bauckhage. 2016. Profiling in Games: Understanding Behavior from Telemetry. In *Social Interaction in Virtual Worlds*.
[21] Gillian Smith and Jim Whitehead. 2010. Analyzing the expressive range of a level generator. In *Proc. of the 2010 Workshop on Proc. Cont. Gen. in Games*. 4.
[22] Gillian Smith, Jim Whitehead, and Michael Mateas. 2010. Tanagra: A mixed-initiative level design tool. In *FDG 2010*. 209–216.
[23] Shoshannah Tekofsky, Pieter Spronck, Martijn Goudbeek, Aske Plaat, and Jaap van den Herik. 2015. Past our prime: A study of age and play style development in battlefield 3. *IEEE TCIAIG* 7, 3 (2015), 292–303.
[24] Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap Van den Herik, and Jan Broersen. 2013. Psyops: Personality assessment through gaming behavior. In *BNAIC 2013*.
[25] Christophe Thiery. 2017. Solarus Engine, an ARPG game engine. http://www.solarus-games.org/. (2017).
[26] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis. 2011. What is procedural content generation?: Mario on the borderline. In *Proc. of the 2nd Int. Workshop on Procedural Content Generation in Games*. 3.
[27] Georgios N Yannakakis and John Hallam. 2009. Real-time game adaptation for optimizing player satisfaction. *IEEE TCIAIG* 1, 2 (2009), 121–133.
[28] Georgios N Yannakakis and Julian Togelius. 2011. Experience-driven procedural content generation. *Affective Computing, IEEE Trans. on* 2, 3 (2011), 147–161.